WebMILE: Democratizing Network Representation Learning at Scale

Yuntian He*
The Ohio State University
he.1773@osu.edu

Yue Zhang*
The Ohio State University
zhang.8016@osu.edu

Saket Gurukar
The Ohio State University
gurukar.1@osu.edu

Srinivasan Parthasarathy
The Ohio State University
srini@cse.ohio-state.edu

ABSTRACT

In recent years, we have seen the success of network representation learning (NRL) methods in diverse domains ranging from computational chemistry to drug discovery and from social network analysis to bioinformatics algorithms. However, each such NRL method is typically prototyped in a programming environment familiar to the developer. Moreover, such methods rarely scale out to large-scale networks or graphs. Such restrictions are problematic to domain scientists or end-users who want to scale a particular NRL method-of-interest on large graphs from their specific domain.

In this work, we present a novel system, WebMILE to democratize this process. WebMILE can scale an unsupervised network embedding method written in the user's preferred programming language on large graphs. It provides an easy-to-use Graphical User Interface (GUI) for the end-user. The user provides the necessary input (embedding method file, graph, required packages information) through a simple GUI, and WebMILE executes the input network embedding method on the given input graph. WebMILE leverages a pioneering multi-level method, MILE (alternatively DistMILE if the user has access to a cluster), that can scale a network embedding method on large graphs. The language agnosticity is achieved through a simple Docker interface. In this demonstration, we will showcase how a domain scientist or end-user can utilize WebMILE to rapidly prototype and learn node embeddings of a large graph in a flexible and efficient manner - ensuring the twin goals of high productivity and high performance.

PVLDB Reference Format:
Yuntian He, Yue Zhang, Saket Gurukar, and Srinivasan Parthasarathy. WebMILE: Democratizing Network Representation Learning at Scale. PVLDB, 15(12): 3718 - 3721, 2022.
doi:10.14778/3554821.3554883

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/heyuntian/WebMILE.

*First two authors contributed equally to this work.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554883

1 INTRODUCTION

Network representation learning (NRL) methods have been utilized in several applications such as molecular property prediction [4], simulation of complex physics [15], construction of disease association network [11] and discover anti-inflammatory agents for COVID-19 [17]. The success of NRL methods has attracted multiple domain scientists and end-users to develop novel network embedding methods that perform well on their selected downstream tasks. However, most NRL methods do not scale on large graphs [12].

Several works [3, 10, 18] have proposed solutions to address the scalability bottleneck of NRL methods. However, these solutions often trade off ease-of-use for optimization. For instance, PytorchBigGraph limits end-users to utilize their proposed first-order proximity preserving embedding method to learn node embeddings. Graphscope [3] requires end-users to adopt Gremlin language for scaling their embedding method on the large graph, while Marius [18] requires users to write their proposed embedding method using a Marius configuration file in Pytorch. Such frameworks are promising and helpful to scale existing methods on large graphs, but they place an additional burden on (or in some cases even restrict) the end-users to execute the developer's custom embedding method in their setting (placing a productivity burden on the end-user).

In this work, we attempt to address the above-mentioned problem by proposing a novel system, WebMILE. WebMILE can execute any custom unsupervised network embedding method on large graphs in a short amount of time without compromising on the quality of the learned embeddings (with respect to the original implementation). WebMILE currently supports the execution of custom network embedding methods written in three programming languages (Python, R, and Java) although in principle it is completely language agnostic and can support any programming environment.

WebMILE design: Our goal with WebMILE is to help domain-scientists either prototype a new method or use an existing method without worrying about scalability issues associated with learning and tuning on large graphs. An additional goal is to facilitate productivity in a programming environment familiar to them. WebMILE in its current avatar is a simple web application (familiar to scientific and social scientists). The current graphic user interface (GUI) helps the end-user navigate the flow of WebMILE with ease. The web application can be deployed locally on the user's server and as a result, the user's graph data does not have to leave his/her device – thereby maintaining the privacy of data. The GUI accepts

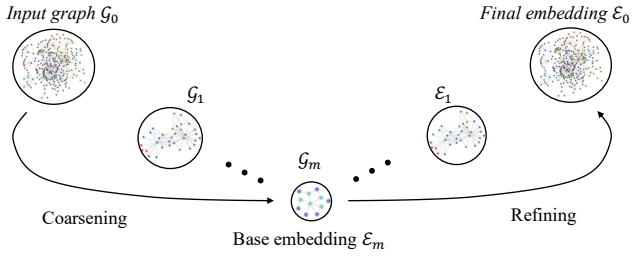


Figure 1: MILE architecture

the necessary code files and the names of the packages required to execute the custom network embedding method.

The next design decision is related to creating WebMILE programming language agnostic. The original principles underlying the development of our multi-level framework - MILE and its distributed variant - DistMILE, ensured that the base embedding method could be completely language agnostic. Specifically, MILE [12], consists of three phases. Given an input graph, in the first phase, it repeatedly coarsens the graph into smaller graphs such that the global structure of the graph is preserved. Next, MILE executes the given network-embedding method on the coarsest graph. In the last phase, MILE learns a graph neural network based refinement model that can learn the embeddings of the input graph from the embeddings of the coarsest graph. DistMILE parallelizes the first and last phases leaving the base embedding phase untouched. In WebMILE we fold these ideas into a Docker environment[1], ensuring that the execution of a user's network embedding is seamless.

The above design enables WebMILE to *learn node embeddings of large graphs in a programming language and network embedding method agnostic manner.*

WebMILE demonstration: We propose a demonstration of WebMILE to data scientists and end-users with different backgrounds to showcase the efficacy of the proposed system on large graphs. We plan to demonstrate:

(1) **WebMILE's simple setup**: We plan to first show how to install the WebMILE web application on a local device. We will also provide the installation instructions and how to start the web server.
(2) **Training on a range of real-world graphs**: Next, we plan to show how to use the WebMILE application to (1) scale a custom network embedding method on a large graph, and (2) incorporate WebMILE with embedding scripts in different programming languages and of different network embedding methodologies.
(3) **Demonstrate the flexibility of WebMILE:** Across multiple commonly leveraged machine learning and data science programming environments through both pre-programmed and on-the-fly embedding technique design, prototyping and deployment in the wild.

2 DEFINITIONS AND PRELIMINARIES

In this section, we briefly present the required definitions and notations and provide a brief overview of MILE.

[1]Docker's platform-as-a-service tool installs relevant packages to execute code.

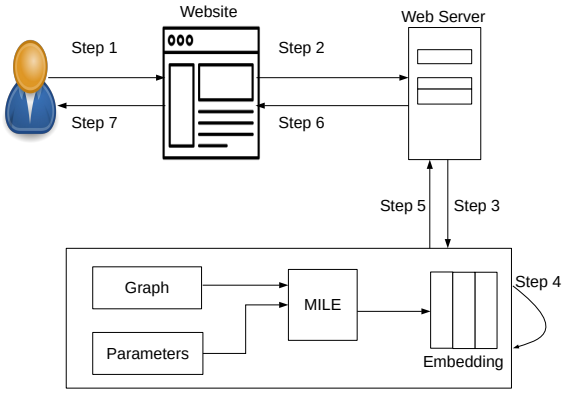


Figure 2: WebMILE training pipeline

Network representation learning [6]: Let $G = (V, E)$ be a graph where $|V|$ number of nodes and $|E|$ number of edges and let d be the embedding dimension, then a network representation learning method learns a d-dimensional representation of the nodes such that similarity in graph space approximates closeness in d-dimensional space.

Several network representation learning methods have been proposed recently. Readers are encouraged to refer to the following surveys for additional details [7, 19]. Most of these methods, however, cannot operate or consume more resources (compute, time) on large graphs. MILE [12] can scale existing network representation learning methods on large graphs in an agnostic manner without compromising on the quality of learned node embeddings.

MILE consists of three phases. Figure 1 shows MILE architecture.

(1) **Coarsening phase**: Given a input graph G (or G_0), MILE repeated coarsens it into series of smaller graphs $G_1, G_2, ...G_m$ such that $|V_0| > |V_1| > ... > |V_m|$ and $|E_0| > |E_1| > ... > |E_m|$. MILE uses a hybrid matching scheme for coarsening. The scheme relies on normalized heavy edge matching (NHEM) [9] and structured equivalence matching [12] techniques.
(2) **Base embedding phase**: In this phase, MILE executes a network embedding method on the coarsest graph G_m and learns the nodes embeddings $\mathcal{E}_m$ of V_m nodes. Note that, in this phase, any network embedding method can be utilized – method agnostic.
(3) **Refinement phase**: In this last phase, MILE learns a graph convolutional network-based refinement model. The model takes input the node embeddings of graph G_i, adjacency matrix of G_{i-1} and the coarse graph mapping between G_i and G_{i-1}. The refinement model then predicts the embeddings of a graph G_{i-1} by incorporating the node embeddings of Graph G_i and graph structure of G_{i-1}. We initialize i to m and then repeatedly predict the node embeddings till we learn the node embeddings of graph G_0 (or G).

3 WEBMILE ARCHITECTURE

Figure 2 depicts the pipeline of the WebMILE. At the high level, the pipeline consists of three stages: i) users upload all the required files

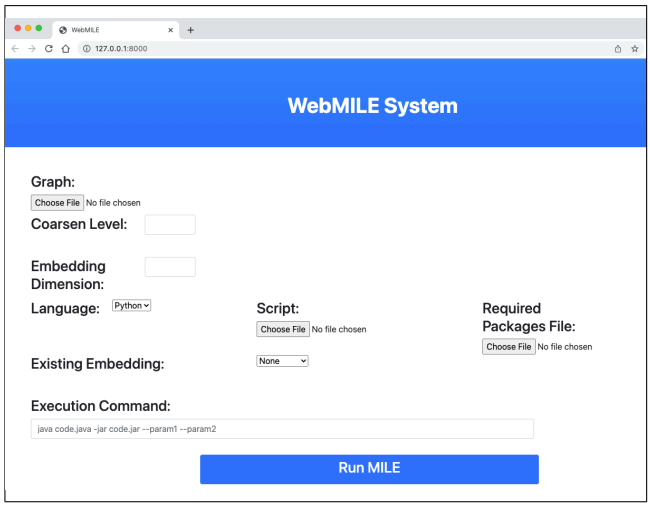


Figure 3: WebMILE web interface

and parameters through the GUI, ii) the webserver then provides all the necessary files and parameters to MILE system and after the MILE execution is finished, iii) the webserver returns the learned embeddings to the users.

At a granular level, the pipeline can be explained as follows: the domain-scientist or end-user visits the web application of WebMILE through a web browser (Step 1). The user is then provided with a GUI shown in Figure 3. Here, the user provides the input graph, the coarsen level, and the embedding dimension of node embedding. We currently support the graph in edge list and metis format [9]. The coarsen level corresponds to the number of times, we coarsen the graph. In our experiments, we observed that increasing the coarsen level results in faster learning of node embeddings, however, higher coarsen levels often result in a reduction in embedding quality. Internally we automate an initial recommendation of coarsen level (based on some initial statistics of the graph that the user wishes to process). The end-user can modulate this value as desired. The end-user then provides the language of the custom embedding script, the custom embedding script file, and the required package file. WebMILE currently supports three programming languages (Python, Java, and R). In the case of python and R, the required package file is expected to be a single text file and each line in the text file should contain a single package name. For Java, the required package file should be the jar file. The user will then enter the execution command and provide the job parameters for executing the embedding method. If the user does not have a custom embedding function, the user can select one of the existing network embedding functions: Deepwalk [13], Node2vec [5], LINE [16], NetMF [14] and GraRep [2]. Clicking on "Run MILE" button will start the execution.

In step 3, we first test the correctness of the input on the web-server. The web-server then creates a docker container and installs the requested software packages. The creation of docker container allows us to execute the model in isolation and avoid package conflicts among different users. After the container creation, we execute MILE on the input graph (step 4). In the base embedding phase of MILE, we utilize the provided custom embedding script to learn node embeddings of the coarsest graph and then utilize the refinement phase of MILE to learn the node embeddings. Once the node embeddings of input graph are learned, we return the embeddings file to the end-user which they can the leverage for downstream analysis.

4 DEMONSTRATION SCENARIOS

Our end-to-end demonstration will i) get the audience familiar with how to set up the WebMILE on a machine and ii) showcase how to use WebMILE to execute customized network embedding methods on large graphs in different scenarios.

4.1 Deployment of WebMILE

WebMILE can be hosted on a wide range of platforms. One can deploy WebMILE on a publicly accessible server that will enable the audience to use it in an easier way. However, such an arrangement will force the end-user to share their data. The end-users might not want to share their data publicly due to privacy concerns. In this case, WebMILE can also be deployed locally so that all data will remain on their own devices.

For local deployment, we will demonstrate how WebMILE can be easily set up by installing Docker and several Python packages. Our graphical user interface is built over a Python-based open-sourced web framework Django [1]. In WebMILE system, our modules for graph coarsening and embedding refinement mostly utilize four Python packages, namely, NumPy, SciPy, Scikit-learn, and TensorFlow. The audience can trivially build a Python interpreter in seconds. In addition to this, Docker is necessary for our framework to run the audience's embedding script in a customized language-agnostic environment.

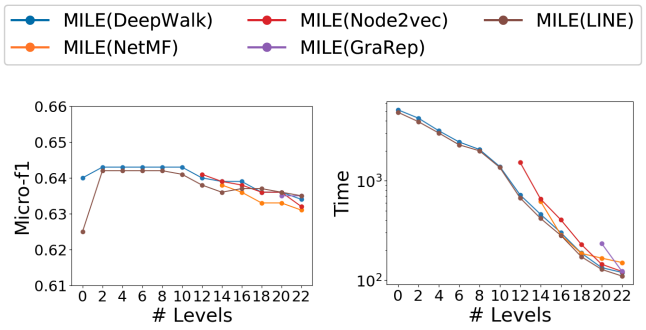
4.2 Training on Large Graphs

After introducing the simplicity and generalizability of WebMILE's deployment, we will demonstrate how to scale the audience's embedding script over large networks. Specifically, the audience will be shown the following scenarios:

Run WebMILE over large datasets: We will first demonstrate how WebMILE can scale the user's customized embedding model over a large dataset. Initially, the user can upload a file of graph data in many common formats such as metis and edge lists. Then the user can specify the parameters of WebMILE and the base embedding model through our graphical interface. To tune the parameters of WebMILE, the user can modify the coarsen level and the dimensionality of embedding. Additionally, the user can tune its customized embedding model by adding arguments in a text box of the execution command. After setting up the data and parameters, WebMILE can return an embedding array in a text file.

To demonstrate the efficacy of WebMILE, we will select one of several real-world datasets (audience choice) and evaluate our framework on either a classification or link prediction task. Our preliminary results show that the system can accelerate the network embedding without compromising the quality. The audience can conduct similar experiments with their specific embedding methods and downstream tasks after our demonstration.

Figure 4 shows the experiments of MILE on the Yelp dataset consisting of 8 million nodes, 40 million edges, and the nodes



(a) Micro-F1 (b) Running Time

Figure 4: Running MILE on Yelp dataset.

have 22 classes. We see that network embedding methods such as Node2vec [5], NetMF [14] and GraRep [2] cannot operate on Yelp dataset. While Deepwalk [13] and LINE [16] can operate on yelp but consumes a significant amount of time. From Figure 4, we see that MILE reduces the running time of DeepWalk from 53 hours to 2 hours with coarsening level 22, while reducing the Micro-F1 score just by 1% (from 0.643 to 0.634).

Language and methodology agnostic embedding: We also plan to demonstrate how WebMILE can perform graph embedding in a language and methodology agnostic manner. The user will upload the embedding script with a dependency list before running WebMILE. Then our system can utilize Docker to build an environment for execution. Details about incorporating an embedding script with WebMILE will be available in our open-source codes. In our demonstration, we will showcase how to run WebMILE with embedding scripts in different languages and using both on-the-fly interactive development and pre-programmed settings.

4.3 Demonstrate the Flexibility of WebMILE

We plan to demonstrate how to utilize WebMILE across multiple machine learning and data science programming environments that are commonly leveraged by domain-scientist and end-users. Here, we will utilize popular pre-programmed network embedding methods in WebMILE and show how to learn the node embeddings of the input graph using those embedding methods.

5 POTENTIAL IMPACT

The adoption of network representation learning methods on large graphs in diverse domains is often limited by the lack of easy-to-use optimized solutions. WebMILE democratizes this process allowing users to flexibly use their custom network representation learning method or adapting a pre-programmed one in a programming language agnostic fashion. Existing optimized solutions require the domain-scientist to either significantly refactor their proposed network embedding method in a format suitable with respect to the optimized solution or require domain-scientist to utilize the optimized solution's embedding method to learn node embeddings. Our proposed system, WebMILE, addresses these problems by proposing an easy-to-use web application that can learn the node embeddings of large graphs efficiently. We believe our proposed system will accelerate the adoption of network representation learning methods in multiple domains, and facilitate rapid prototyping of new methods that require learning of node embeddings on large graphs. We are currently working on improving the simplistic interface to enhance usability in resource-constrained environments (training on the edge). We are also currently extending the infrastructure to take advantage of additional parallelization optimizations as discussed in DistMILE [8] as this will allow users to train their custom network embedding methods on even larger graphs in a shorter amount of time.

ACKNOWLEDGMENTS

This material is supported by the National Science Foundation (NSF) under grants OAC-2018627, CCF-2028944, and CNS-2112471. Any opinions, findings, and conclusions in this material are those of the author(s) and may not reflect the views of the respective funding agency.

REFERENCES

[1] Django. https://www.djangoproject.com
[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning Graph Representations with Global Structural Information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 891–900.
[3] Wenfei Fan et al. 2021. GraphScope: a unified engine for big graph processing. *VLDB* (2021).
[4] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272.
[5] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 855–864.
[6] Saket Gurukar, Priyesh Vijayan, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, Anasua Mitra, Vedang Patel, et al. 2019. Network representation learning: Consolidation and renewed bearing. *arXiv preprint arXiv:1905.00987* (2019).
[7] William L Hamilton. 2020. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14, 3 (2020), 1–159.
[8] Yuntian He, Saket Gurukar, Pouya Kousha, Hari Subramoni, Dhabaleswar K Panda, and Srinivasan Parthasarathy. 2021. DistMILE: A Distributed Multi-Level Framework for Scalable Graph Embedding. In *HiPC*. IEEE, 282–291.
[9] George Karypis and Vipin Kumar. 1997. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. (1997).
[10] Adam Lerer et al. 2019. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems* 1 (2019), 120–131.
[11] Michelle M Li, Kexin Huang, and Marinka Zitnik. 2021. Representation learning for networks in biology and medicine: advancements, challenges, and opportunities. *arXiv preprint arXiv:2104.04883* (2021).
[12] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. 2021. MILE: A Multi-Level Framework for Scalable Graph Embedding. *Proceedings of the International AAAI Conference on Web and Social Media* 15, 1 (2021), 361–372.
[13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 701–710.
[14] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2vec. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. 459–467.
[15] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. 2020. Graph neural networks in particle physics. *Machine Learning: Science and Technology* (2020).
[16] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. 1067–1077.
[17] Xiaoqi Wang et al. 2021. DeepR2cov: deep representation learning on heterogeneous drug networks to discover anti-inflammatory agents for COVID-19. *Briefings in Bioinformatics* (2021).
[18] Anze Xie et al. 2021. Demo of marius: a system for large-scale graph embeddings. *VLDB* (2021).
[19] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. Network representation learning: A survey. *IEEE transactions on Big Data* (2018).